Copy Fail: 732 Bytes to Root on Every Major Linux Distribution.

Xint Code disclosed CVE-2026-31431, an authentication bypass in the Linux kernel's AEAD subsystem. It is a logic bug in a single 732-byte script. A 732-byte PoC gets root on Ubuntu, Amazon Linux, RHEL, SUSE.

Apr 29, 2026

Copy Fail: 732 Bytes to Root on Every Major Linux Distribution.

Xint Code

Contents

- What Kernel Logic Bugs Can Do Different
- The Root Cause: Page Cache Pages in the Writable Scatterlist
- The Trigger: Authencesn's Scratch Write
- How This Happened
- The Exploit
- The Demo
- The Fix
- Remediation
- Coordinated Disclosure Timeline
- How We Found It

Copy Fail. 732 Bytes to Root on Every Major Linux Distribution.

Copy Fail (CVE-2026-31431) is a logic bug in the Linux kernel's authencesn cryptographic template. It lets an unprivileged local user trigger a deterministic, controlled 4-byte write into the page cache of any readable file in the system. A single 732-byte Python script run edit a setuid binary and obtain root on Ubuntu 24.04 LTS distributions shipped since 2017.

The kernel never marks the corrupted page dirty for writeback, so the file on disk stays unchanged and integrity checks against the package manifest miss the modification. However, the page cache is what actually gets read when executing the file, so the corrupted in-memory version is immediately visible to every system wide. A local unprivileged user can turn this into root by corrupting the page cache of a setuid binary. The same primitive bypasses container boundaries because the page cache is shared across the host kernel.

The entire exploit is a stock Python script using standard AF_ALG and Unix Socket bindings. It works on Ubuntu, Amazon Linux 2023, RHEL 10, and SUSE. No per-distro offsets, No recompilation. No version checks in the exploit.

The entire exploit is a stock Python script using standard library modules (os, socket, zlib). It requires Python 3.10+ for os.splice. No compiled artifacts, no dependencies, no offsets.

Stealthy. The write bypasses the ordinary VFS write path. The corrupted page is never marked dirty by the kernel's writeback machinery. Standard file integrity checks against the package manifest miss it. The on-disk binary stays clean. Only the in-memory page cache is corrupted.

Container-transparent. Impact. The page cache is shared across all processes on a system, including across container boundaries. Copy Fail is not just a local privilege escalation. It is a container escape primitive and a Kubernetes node compromise vector (Part 2).

The Root Cause: Page Cache Pages in the Writable Scatterlist

AF_ALG is a socket type that exposes the kernel's crypto subsystem to unprivileged userspace. A user can open a socket, bind to any AEAD (authenticated-encryption with associated data) algorithm, set a key, provide encryption or decryption or arbitrary data. The cryptographic work happens in place.

splice() is a primitive underlying the Linux sendfile(2). It moves data between file descriptors and pipes without copying, passing page cache pages by reference. For AEAD ciphers, the input is a scatterlist of pages. If splice() feeds the socket's input scatterlist holds direct references to the kernel's cached pages of a file. The pages are attached to the scatterlist as entries of the same physical pages that back every read(), mmap(), and exec() of that file.

AEAD ciphers also write to their output scatterlist. authencesn, the IPsec ESN (sphenic) wrapper, decrypts aes(cbc), aes(ctr), rfc4543(gcm) sets up the scatterlist as in-place decrypting the same scatterlist twice as both input and output for the crypto algorithm.

The AD and ciphertext data are byte-copied from the input scatterlist into the output buffer via memcpy_sglist. This is a real copy. The page cache pages are not touched. But authencesn then performs a scratch write to the output scatterlist, is not copied. The kernel restores the scatterlist entries for the tag to point back to the original page cache pages. The scratch write goes to the page cache.

```
input SGL:  [ AD ] [ CT ] [ TAG ]
            [ AD ] [ CT ] [ TAG ]
            [ AD ] [ CT ] [ TAG ]
                          ^ scratch_buf[0:4] authenesn: page cache page
output SGL: [ AD ] [ CT ] [ TAG ]
```

The output scatterlist now has two regions: the user's recvmsg buffer (containing correct AD and ciphertext), followed by the chained tag pages, still referencing the original page cache pages of the target file. The kernel's AEAD layer then reads tag bytes, pushing to the head of this combined chain:

```
crypto_aead_decrypt(req)
  authenesn_decrypt_tail(req, PAGE_ACCEPT)
    scatterwalk_map_and_copy(tmp, dst, 4, 4, 1);   // save
    scatterwalk_ffwd(dst, dst, assoclen + cryptlen);
    sg_chain(rctx->sg, 2, dst);                    // chain page cache tag
    aead_request_set_tfb(req, NULL);               // NULL tfb
    crypto_aead_decrypt(req);                      // AES-CBC decrypt
    scatterwalk_map_and_copy(tmp, dst, 4, 4, 0);   // restore
```

This is place dropping is the root cause of the vulnerability. It places page cache pages in a writable scatterlist, expected from the legitimate write region by requiring more than an offset boundary. The legitimate operation (IPsec algorithm will confirm) he writes to the intended destination, but nothing in the architecture this, and nothing stops authencesn from doing it.

Unfortunately, one AEAD algorithm breaks this model.

The Trigger: authencesn's Scratch Write

The kernel's AEAD API defines a clear output contract for decryption: the destination buffer receives AD | plaintext, exactly authsize bytes. authencesn breaks this.

authencesn is an AEAD wrapper used by IPsec for Extended Sequence Number (ESN) support. IPsec uses 64-bit sequence numbers split into a high half (seq_no_hi) and a low half (seq_no_lo). authencesn's job is to MAC the full sequence, but only transmit the low 4 bytes. For HMAC computation, authencesn needs to rearrange these bytes: seq_no_hi at the front of the hash input, seqno_lo appended at the end.

It performs this rearrangement by using the caller's destination buffer as scratch space. In crypto_authenc_decrypt:

```
crypto_authenc_decrypt_tail(req, PAGE_ACCEPT);
  authenc_ahash_fn = crypto_ahash_spawn_ahash(ctx->auth);
  ahash = crypto_fork_ahash(authenc_ahash_fn, req->dst, req->cryptlen, 0, 1);
  ahash_request_set_tfb(ahash_req, req->dst);
  ahash_request_set_crypt(ahash_req, req->dst, hdr, 0);
  crypto_ahash_digest(ahash_req);
```

The first two calls shuffle the IPsec ESN within the AEAD region, a temporary modification that gets restored. The third call writes 4 bytes of after authen = hmac(K, seq_no_hi || AD || ciphertext) to offset cryptlen in req->dst. The destination is the user's recvmsg buffer.

authencesn's AEAD wrapper used by IPsec for Extended Sequence Number (ESN) support. IPsec uses 64-bit sequence numbers split into a high half (seq_no_hi), bytes 0–3 of the AAD, and a low half (seqno_lo, bytes 4–7). The wire format carries only seqno_lo, seqno_hi is implicit. For HMAC computation, authencesn needs to rearrange these bytes: seqno_hi at the front of the hash input, seqno_lo appended at the end.

It performs this rearrangement by using the caller's destination buffer as scratch space. In crypto_authenc_decrypt:

```
crypto_authenc_decrypt_tail(req, PAGE_ACCEPT);
  authenc_ahash_fn = crypto_ahash_spawn_ahash(ctx->auth);
  ahash = crypto_fork_ahash(authenc_ahash_fn, req->dst, req->cryptlen, 0, 1);
  ahash_request_set_tfb(ahash_req, req->dst);
  ahash_request_set_crypt(ahash_req, req->dst, hdr, 0);
  crypto_ahash_digest(ahash_req);
```

The first two calls shuffle the ESN bytes within the AAD region, a temporary modification that gets restored. The third call writes 4 bytes of after authen = hmac(K, seqno_hi || AD || ciphertext) to offset cryptlen in req->dst. The destination is the user's recvmsg buffer. For HMAC computation, authencesn needs to rearrange these bytes: seqno_hi at the front of the hash input, seqno_lo appended at the end.

authencesn's scratch write. authencesn writes the HMAC tag to dst[cryptlen], with seqno_hi at dst[assoclen + cryptlen]. These are the positions immediately after the AD and ciphertext. For a normal operation, authencesn's scratch write lands in the caller's recvmsg buffer. For AF_ALG, the caller's buffer is the kernel's page cache. authencesn writes 4 bytes of authen = hmac(K, seqno_hi || AD || ciphertext) to offset cryptlen in req->dst. The destination is the user

Grit Code is a security research tool built for this workflow: a researcher identifies the attack surface, KC analyzes it.

Next: "From PoC to Hosts", how Copy Fail escapes every major cloud Kubernetes platform.

Share article

Write your description body here.

Email

Enter your email

Submit